

SPARQL – Syntax und Intuition

Sebastian Rudolph

Institut AIFB · Universität Karlsruhe

Grundlagen Semantic Web (WS09/10)

Seminar für Computerlinguistik, Universität Heidelberg

<http://semantic-web-grundlagen.de>

Foliensatz von M. Krötzsch. Die nichtkommerzielle Vervielfältigung, Verbreitung und Bearbeitung dieser Folien ist zulässig (→ Lizenzbestimmungen CC-BY-NC).

- 1 Einleitung und Motivation
- 2 Einfache SPARQL-Anfragen
- 3 Komplexe Graph-Muster in SPARQL
- 4 Filter in SPARQL
- 5 Ausgabeformate in SPARQL
- 6 Modifikatoren in SPARQL
- 7 Zusammenfassung und Ausblick

- 1 Einleitung und Ausblick
- 2 XML und URIs
- 3 Einführung in RDF
- 4 RDF Schema
- 5 Logik – Grundlagen
- 6 Semantik von RDF(S)
- 7 OWL – Syntax und Intuition
- 8 OWL – Semantik und Reasoning
- 9 OWL 2 – Syntax und Semantik
- 10 **SPARQL – Syntax und Intuition** (→ Webseite dieser Vorlesung)
- 11 Semantik von SPARQL
- 12 Konjunktive Anfragen und Regelsprachen

Wie kann auf in RDF oder OWL spezifizierte Informationen zugegriffen werden?

Wie kann auf in RDF oder OWL spezifizierte Informationen zugegriffen werden?

Abfrage von Informationen in RDF(S)

- Einfache Folgerung
- RDF-Folgerung
- RDFS-Folgerung

„Folgt ein bestimmter RDF-Graph aus einem gegebenen?“

Wie kann auf in RDF oder OWL spezifizierte Informationen zugegriffen werden?

Abfrage von Informationen in RDF(S)

- Einfache Folgerung
- RDF-Folgerung
- RDFS-Folgerung

„Folgt ein bestimmter RDF-Graph aus einem gegebenen?“

Abfrage von Informationen in OWL

- Logisches Schließen

„Folgt eine Subklassen-Beziehung aus einer OWL-Ontologie?“

„Welches sind die Instanzen einer Klasse einer OWL-Ontologie?“

Genügen OWL und RDF nicht?

Selbst OWL ist als Anfragesprache oft zu schwach

- „Welche Zeichenketten in deutscher Sprache sind in der Ontologie angegeben?“
- „Welche Propertys verbinden zwei bestimmte Individuen?“
- „Welche Paare von Personen haben eine gemeinsames Elternteil?“

~> weder in RDF noch in OWL ausdrückbar.

Genügen OWL und RDF nicht?

Selbst OWL ist als Anfragesprache oft zu schwach

- „Welche Zeichenketten in deutscher Sprache sind in der Ontologie angegeben?“
- „Welche Propertys verbinden zwei bestimmte Individuen?“
- „Welche Paare von Personen haben eine gemeinsames Elternteil?“

↪ weder in RDF noch in OWL ausdrückbar.

Anforderungen:

- Große Ausdruckstärke zur Beschreibung der gefragten Information
- Möglichkeiten zur Formatierung, Einschränkung und Manipulation der Ergebnisse

Agenda für diese und die folgende Vorlesung:

diese Session:

- Grundlagen der RDF-Anfragesprache SPARQL

nächste Session:

- Semantik der RDF-Anfragesprache SPARQL
- Konjunktive Anfragen für OWL

- 1 Einleitung und Motivation
- 2 Einfache SPARQL-Anfragen**
- 3 Komplexe Graph-Muster in SPARQL
- 4 Filter in SPARQL
- 5 Ausgabeformate in SPARQL
- 6 Modifikatoren in SPARQL
- 7 Zusammenfassung und Ausblick

SPARQL (sprich engl. *sparkle*) steht für
SPARQL Protocol And RDF Query Language

- W3C-Spezifikation kurz vor Standardisierung
- Anfragsprache zur *Abfrage von Instanzen aus RDF-Dokumenten*
- schon heute große praktische Bedeutung

Teile der SPARQL-Spezifikation

- Anfragesprache: Thema dieser Vorlesung
- Ergebnisformat: Darstellung von Ergebnissen in XML
- Anfrageprotokoll: Übermittlung von Anfragen und Ergebnissen

Eine einfache Beispielanfrage:

```
PREFIX ex: <http://example.org/>
SELECT ?titel ?autor
WHERE
{ ?buch      ex:VerlegtBei    <http://springer.com/Verlag> .
  ?buch      ex:Titel         ?titel .
  ?buch      ex:Autor         ?autor . }
```

Eine einfache Beispielanfrage:

```
PREFIX ex: <http://example.org/>
SELECT ?titel ?autor
WHERE
{ ?buch      ex:VerlegtBei    <http://springer.com/Verlag> .
  ?buch      ex:Titel         ?titel .
  ?buch      ex:Autor         ?autor . }
```

- Hauptbestandteil ist ein **Anfragemuster** (WHERE)
 - ↪ Anfragemuster verwenden die Turtle-Syntax für RDF
 - ↪ Muster dürfen Variablen enthalten (?variable)
- **Kurzschreibweisen** für URIs möglich (PREFIX)
- Anfrageergebnis durch **Auswahl von Variablen** (SELECT)

Beispiel RDF-Dokument:

```
@prefix ex: <http://example.org/> .  
ex:SemanticWeb ex:VerlegtBei <http://springer.com/Verlag> ;  
               ex:Titel      "Semantic Web - Grundlagen" ;  
               ex:Autor      ex:Hitzler, ex:Kröttsch,  
                             ex:Rudolph, ex:Sure .
```

Ergebnis der Anfrage: Tabelle mit einer Zeile je Ergebnis

titel	autor
"Semantic Web - Grundlagen"	http://example.org/Hitzler
"Semantic Web - Grundlagen"	http://example.org/Kröttsch
"Semantic Web - Grundlagen"	http://example.org/Rudolph
"Semantic Web - Grundlagen"	http://example.org/Sure

Die grundlegenden Anfragemuster sind **einfache Graph-Muster**

- Menge von RDF-Tripeln in Turtle-Syntax
- Turtle-Abkürzungen (mittels `,` und `;`) zulässig
- Variablen werden durch `?` oder `$` gekennzeichnet (`?variable` hat gleiche Bedeutung wie `$variable`)
- Variablen zulässig als Subjekt, Prädikat oder Objekt

Einfache Graph-Muster

Die grundlegenden Anfragemuster sind **einfache Graph-Muster**

- Menge von RDF-Tripeln in Turtle-Syntax
- Turtle-Abkürzungen (mittels `,` und `;`) zulässig
- Variablen werden durch `?` oder `$` gekennzeichnet (`?variable` hat gleiche Bedeutung wie `$variable`)
- Variablen zulässig als Subjekt, Prädikat oder Objekt

Zulässig \neq lesbar:

```
PREFIX ex:    <http://example.org/>
SELECT $rf456df ?_AIFB WHERE { ?ef3a_3 ex:VerlegtBei
<http://springer.com/Verlag> . ?ef3a_3 ex:Titel
    ?rf456df . $ef3a_3      ex:Autor ?_AIFB . }
```

(semantisch äquivalent zur vorherigen Anfrage)

Was bedeuten leere Knoten in SPARQL?

Leere Knoten in Anfragemustern:

- Zulässig als Subjekt oder Objekt
- ID beliebig, aber niemals gleiche ID mehrfach pro Anfrage
- Verhalten sich wie Variablen, die nicht ausgewählt werden können

Was bedeuten leere Knoten in SPARQL?

Leere Knoten in Abfragemustern:

- Zulässig als Subjekt oder Objekt
- ID beliebig, aber niemals gleiche ID mehrfach pro Abfrage
- Verhalten sich wie Variablen, die nicht ausgewählt werden können

Leere Knoten in Ergebnissen:

- Platzhalter für unbekannte Elemente
- IDs beliebig, aber eventuell an andere Ergebnisteile gebunden:

subj	wert
_:a	"zum"
_:b	"Beispiel"

subj	wert
_:y	"zum"
_:g	"Beispiel"

subj	wert
_:z	"zum"
_:z	"Beispiel"

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
@prefix ex:  <http://example.org/> .  
ex:bsp1    ex:p    "test" .  
ex:bsp2    ex:p    "test"^^xsd:string .  
ex:bsp3    ex:p    "test"@de .  
ex:bsp4    ex:p    "42"^^xsd:integer .
```

Was liefert eine Anfrage mit folgendem Muster?

```
{      ?subject    <http://example.org/p>    "test" . }
```

Datentypen

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/> .
ex:bsp1 ex:p "test" .
ex:bsp2 ex:p "test"^^xsd:string .
ex:bsp3 ex:p "test"@de .
ex:bsp4 ex:p "42"^^xsd:integer .
```

Was liefert eine Anfrage mit folgendem Muster?

```
{      ?subject <http://example.org/p> "test" . }
```

⇒ ex:bsp1 als einziges Ergebnis

⇒ genaue Übereinstimmung der Datentypen gefordert

Aber: Abkürzung für Zahlenwerte möglich

```
{      ?subject <http://example.org/p> 42 . }
```

⇒ Datentyp wird aus syntaktischer Form bestimmt

(xsd:integer, xsd:decimal, oder xsd:double)

- 1 Einleitung und Motivation
- 2 Einfache SPARQL-Anfragen
- 3 Komplexe Graph-Muster in SPARQL**
- 4 Filter in SPARQL
- 5 Ausgabeformate in SPARQL
- 6 Modifikatoren in SPARQL
- 7 Zusammenfassung und Ausblick

Einfache Graph-Muster können durch {...} gruppiert werden.

Beispiel:

```
PREFIX ex: <http://example.org/>
SELECT ?titel ?autor
WHERE
  { { ?buch      ex:VerlegtBei <http://springer.com/Verlag>
    ?buch      ex:Titel      ?titel . }
    { }
    ?buch      ex:Autor      ?autor .
  }
```

⇒ Sinnvoll erst bei Verwendung zusätzlicher Konstruktoren

Das Schlüsselwort `OPTIONAL` erlaubt die Angabe optionaler Teile eines Musters.

Beispiel:

```
{ ?buch      ex:VerlegtBei  <http://springer.com/Verlag> .  
  OPTIONAL { ?buch      ex:Titel      ?titel . }  
  OPTIONAL { ?buch      ex:Autor      ?autor . }  
}
```

Das Schlüsselwort `OPTIONAL` erlaubt die Angabe optionaler Teile eines Musters.

Beispiel:

```
{ ?buch      ex:VerlegtBei  <http://springer.com/Verlag> .  
  OPTIONAL { ?buch      ex:Titel      ?titel . }  
  OPTIONAL { ?buch      ex:Autor      ?autor . }  
}
```

⇒ Teile eines Anfrageergebnisses können **ungebunden** sein:

buch	titel	autor
http://example.org/buch1	"Titel1"	http://example.org/autor1
http://example.org/buch2	"Titel2"	
http://example.org/buch3	"Titel3"	_:a
http://example.org/buch4		_:a
http://example.org/buch5		

Das Schlüsselwort `UNION` erlaubt die Angabe alternativer Teile eines Musters.

Beispiel:

```
{ ?buch      ex:VerlegtBei    <http://springer.com/Verlag> .  
  { ?buch      ex:Autor        ?autor . } UNION  
  { ?buch      ex:Verfasser    ?autor . }  
}
```

↪ Ergebnis entspricht Vereinigung der Ergebnisse mit einer der beiden Bedingungen

Anm.: Gleiche Variablennamen in beiden Teilen von `UNION` beeinflussen sich nicht

Kombination von Optionen und Alternativen (1)

Wie sind Kombinationen von OPTIONAL und UNION zu verstehen?

```
{ ?buch      ex:VerlegtBei    <http://springer.com/Verlag> .  
  { ?buch      ex:Autor        ?autor . } UNION  
  { ?buch      ex:Verfasser    ?autor . } OPTIONAL  
  { ?autor     ex:Nachname     ?name . }  
}
```

- Vereinigung zweier Muster mit angefügtem optionalem Muster
oder
- Vereinigung zweier Muster, wobei das zweite einen optionalen Teil hat?

Kombination von Optionen und Alternativen (1)

Wie sind Kombinationen von OPTIONAL und UNION zu verstehen?

```
{ ?buch      ex:VerlegtBei    <http://springer.com/Verlag> .  
  { ?buch      ex:Autor        ?autor . } UNION  
  { ?buch      ex:Verfasser    ?autor . } OPTIONAL  
  { ?autor     ex:Nachname     ?name . }  
}
```

- Vereinigung zweier Muster mit angefügtem optionalem Muster
oder
- Vereinigung zweier Muster, wobei das zweite einen optionalen Teil hat?

⇒ Erste Interpretation korrekt:

```
{ ?buch      ex:VerlegtBei    <http://springer.com/Verlag> .  
  { { ?buch      ex:Autor        ?autor . } UNION  
    { ?buch      ex:Verfasser    ?autor . }  
  } OPTIONAL { ?autor     ex:Nachname     ?name . }  
}
```

Kombination von Optionen und Alternativen (2)

Allgemeine Regeln

- `OPTIONAL` bezieht sich immer auf genau ein gruppierendes Muster rechts davon.
- `OPTIONAL` und `UNION` sind gleichwertig und beziehen sich auf jeweils alle links davon stehenden Ausdrücke (*linksassoziativ*)

Beispiel:

```
{ {s1 p1 o1} OPTIONAL {s2 p2 o2} UNION {s3 p3 o3}  
  OPTIONAL {s4 p4 o4} OPTIONAL {s5 p5 o5}  
}
```

Kombination von Optionen und Alternativen (2)

Allgemeine Regeln

- **OPTIONAL** bezieht sich immer auf genau ein gruppierendes Muster rechts davon.
- **OPTIONAL** und **UNION** sind gleichwertig und beziehen sich auf jeweils alle links davon stehenden Ausdrücke (*linksassoziativ*)

Beispiel:

```
{ {s1 p1 o1} OPTIONAL {s2 p2 o2} UNION {s3 p3 o3}
  OPTIONAL {s4 p4 o4} OPTIONAL {s5 p5 o5}
}
```

bedeutet

```
{ { { { {s1 p1 o1} OPTIONAL {s2 p2 o2}
      } UNION {s3 p3 o3}
    } OPTIONAL {s4 p4 o4}
  } OPTIONAL {s5 p5 o5}
}
```

- 1 Einleitung und Motivation
- 2 Einfache SPARQL-Anfragen
- 3 Komplexe Graph-Muster in SPARQL
- 4 Filter in SPARQL**
- 5 Ausgabeformate in SPARQL
- 6 Modifikatoren in SPARQL
- 7 Zusammenfassung und Ausblick

Wozu Filter?

Viele Anfragen sind auch mit komplexen Graph-Mustern nicht möglich:

- „Welche Personen sind zwischen 18 und 23 Jahre alt?“
- „Der Nachname welcher Personen enthält einen Bindestrich?“
- „Welche Texte in deutscher Sprache sind in der Ontologie angegeben?“

⇒ **Filter** als allgemeiner Mechanismus für solche Ausdrucksmittel

Beispiel:

```
PREFIX ex: <http://example.org/>
SELECT ?buch WHERE
{ ?buch ex:VerlegtBei <http://springer.com/Verlag> .
  ?buch ex:Preis      ?preis
  FILTER (?preis < 35)
}
```

- Schlüsselwort `FILTER`, gefolgt von Filterausdruck in Klammern
- Filterbedingungen liefern Wahrheitswerte (und ev. auch Fehler)
- Viele Filterfunktionen nicht durch RDF spezifiziert
 ↪ Funktionen teils aus XQuery/XPath-Standard für XML übernommen

Vergleichoperatoren: `<`, `=`, `>`, `<=`, `>=`, `!=`

- Vergleich von Datenliteralen gemäß der jeweils *natürlichen* Reihenfolge
- Unterstützung für numerische Datentypen, `xsd:dateTime`, `xsd:string` (alphabetische Ordnung), `xsd:Boolean` (`1 > 0`)
- für andere Typen und sonstige RDF-Elemente nur `=` und `!=` verfügbar
- kein Vergleich von Literalen inkompatibler Typen (z.B. `xsd:string` und `xsd:integer`)

Filterfunktionen: Vergleiche und Arithmetik

Vergleichoperatoren: `<`, `=`, `>`, `<=`, `>=`, `!=`

- Vergleich von Datenliteralen gemäß der jeweils *natürlichen* Reihenfolge
- Unterstützung für numerische Datentypen, `xsd:dateTime`, `xsd:string` (alphabetische Ordnung), `xsd:Boolean` (`1 > 0`)
- für andere Typen und sonstige RDF-Elemente nur `=` und `!=` verfügbar
- kein Vergleich von Literalen inkompatibler Typen (z.B. `xsd:string` und `xsd:integer`)

Arithmetische Operatoren: `+`, `-`, `*`, `/`

- Unterstützung für numerische Datentypen
- Verwendung zur Kombination von Werten in Filterbedingungen

Bsp.: `FILTER (?gewicht / (?groesse * ?groesse) >= 25)`

Filterfunktionen: Spezialfunktionen für RDF (1)

SPARQL unterstützt auch **RDF-spezifische Filterfunktionen**:

<code>BOUND (A)</code>	<code>true</code> falls A eine gebundene Variable ist
<code>isURI (A)</code>	<code>true</code> falls A eine URI ist
<code>isBLANK (A)</code>	<code>true</code> falls A ein leerer Knoten ist
<code>isLITERAL (A)</code>	<code>true</code> falls A ein RDF-Literal ist
<code>STR (A)</code>	lexikalische Darstellung (<code>xsd:string</code>) von RDF-Literalen oder URIs
<code>LANG (A)</code>	Sprachcode eines RDF-Literals (<code>xsd:string</code>) oder leerer String falls kein Sprachcode
<code>DATATYPE (A)</code>	Datentyp-URI eines RDF-Literals (<code>xsd:string</code> bei ungetypten Literalen ohne Sprachangabe)

Filterfunktionen: Spezialfunktionen für RDF (2)

Weitere RDF-spezifische Filterfunktionen:

<code>sameTERM (A, B)</code>	true, falls A und B dieselben RDF-Terme sind.
<code>langMATCHES (A, B)</code>	true, falls die Sprachangabe A auf das Muster B passt
<code>REGEX (A, B)</code>	true, falls in der Zeichenkette A der reguläre Ausdruck B gefunden werden kann

Beispiel:

```
PREFIX ex: <http://example.org/>
SELECT ?buch WHERE
{ ?buch      ex:Rezension  ?text .
  FILTER ( langMATCHES ( LANG(?text), "de") )
}
```

Filterfunktionen: Boolesche Operatoren

Filterbedingungen können mit **Booleschen Operatoren** verknüpft werden: `&`, `|`, `!`

Teilweise auch durch Graph-Muster ausdrückbar:

- Konjunktion entspricht Angaben mehrerer Filter
- Disjunktion entspricht Anwendung von Filtern in alternativen Mustern

- 1 Einleitung und Motivation
- 2 Einfache SPARQL-Anfragen
- 3 Komplexe Graph-Muster in SPARQL
- 4 Filter in SPARQL
- 5 Ausgabeformate in SPARQL**
- 6 Modifikatoren in SPARQL
- 7 Zusammenfassung und Ausblick

Ausgabeformatierung mit SELECT

Bisher waren alle Ergebnisse Tabellen: Ausgabeformat `SELECT`

Syntax: `SELECT <Variablenliste> oder SELECT *`

Vorteil

Einfache sequentielle Abarbeitung von Ergebnissen

Nachteil

Struktur/Beziehungen der Objekte im Ergebnis nicht offensichtlich

Ausgabeformatierung mit CONSTRUCT

Kodierung von Ergebnissen in RDF-Graphen: Ausgabeformat CONSTRUCT

Syntax: CONSTRUCT <RDF-Schablone in Turtle>

```
PREFIX ex: <http://example.org/>
CONSTRUCT { ?person ex:mailbox ?email .
             ?person ex:telefon ?telefon . }
WHERE { ?person ex:email ?email .
        ?person ex:tel ?telefon . }
```


Ausgabeformatierung mit CONSTRUCT

Kodierung von Ergebnissen in RDF-Graphen: Ausgabeformat CONSTRUCT

Syntax: CONSTRUCT <RDF-Schablone in Turtle>

```
PREFIX ex: <http://example.org/>
CONSTRUCT { ?person ex:mailbox ?email .
             ?person ex:telefon ?telefon . }
WHERE { ?person ex:email ?email .
        ?person ex:tel ?telefon . }
```

Vorteil

Strukturiertes Ergebnis mit Beziehungen zwischen Elementen

Nachteile

- Sequentielle Abarbeitung von Ergebnissen erschwert
- Keine Behandlung von ungebundenen Variablen

SPARQL unterstützt zwei weitere Ausgabeformate:

- ASK prüft nur, ob es Ergebnisse gibt, keine Parameter
- DESCRIBE (informativ) liefert zu jeder gefundenen URI eine RDF-Beschreibung (anwendungsabhängig)

- 1 Einleitung und Motivation
- 2 Einfache SPARQL-Anfragen
- 3 Komplexe Graph-Muster in SPARQL
- 4 Filter in SPARQL
- 5 Ausgabeformate in SPARQL
- 6 Modifikatoren in SPARQL**
- 7 Zusammenfassung und Ausblick

Wozu Modifikatoren?

Bisher nur grundsätzliche Formatierungseinstellungen für Ergebnisse:

- Wie kann man definierte Teile der Ergebnismenge abfragen?
- Wie werden Ergebnisse geordnet?
- Können wiederholte Ergebniszeilen sofort entfernt werden?

⇒ **Modifikatoren** der Lösungssequenz (*solution sequence modifiers*)

Ergebnisse sortieren

Sortierung von Ergebnissen mit Schlüsselwort ORDER BY

```
SELECT ?buch, ?preis  
WHERE { ?buch <http://example.org/Preis> ?preis . }  
ORDER BY ?preis
```

Ergebnisse sortieren

Sortierung von Ergebnissen mit Schlüsselwort ORDER BY

```
SELECT ?buch, ?preis
WHERE { ?buch <http://example.org/Preis> ?preis . }
ORDER BY ?preis
```

- Sortierung wie bei Filter-Vergleichoperatoren,
- Sortierung von URIs alphabetisch als Zeichenketten
- Reihenfolge zwischen unterschiedlichen Arten von Elementen:
Ungebundene Variable < leere Knoten < URIs < RDF-Literale
- nicht jede Möglichkeit durch Spezifikation definitert

Ergebnisse sortieren

Sortierung von Ergebnissen mit Schlüsselwort ORDER BY

```
SELECT ?buch, ?preis
WHERE { ?buch <http://example.org/Preis> ?preis . }
ORDER BY ?preis
```

- Sortierung wie bei Filter-Vergleichoperatoren,
- Sortierung von URIs alphabetisch als Zeichenketten
- Reihenfolge zwischen unterschiedlichen Arten von Elementen:
Ungebundene Variable < leere Knoten < URIs < RDF-Literale
- nicht jede Möglichkeit durch Spezifikation definitert

Weitere mögliche Angaben:

- ORDER BY DESC(?preis): **absteigend**
- ORDER BY ASC(?preis): **aufsteigend**, Voreinstellung
- ORDER BY DESC(?preis), ?titel: **hierarchische**
Ordnungskriterien

Einschränkung der Ergebnismenge:

- LIMIT: maximale Anzahl von Ergebnissen (Tabellenzeilen)
- OFFSET: Position des ersten gelieferten Ergebnisses
- SELECT DISTINCT: Entfernung von doppelten Tabellenzeilen

```
SELECT DISTINCT ?buch, ?preis
WHERE { ?buch <http://example.org/Preis> ?preis . }
ORDER BY ?preis LIMIT 5 OFFSET 25
```

↪ LIMIT und OFFSET nur mit ORDER BY sinnvoll!

Reihenfolge bei Abarbeitung von Modifikatoren:

- 1 Sortierung gemäß `ORDER BY`
- 2 Entfernung der nicht ausgewählten Variablen
- 3 Entfernung doppelter Ergebnisse (`DISTINCT`)
- 4 Entfernung der ersten `OFFSET` Ergebnisse
- 5 Entfernung aller Ergebnisse bis auf `LIMIT`

↪ Sortierung auch nach nicht ausgewählten Variablen möglich

↪ `ORDER BY` nicht nur für `SELECT` relevant

- 1 Einleitung und Motivation
- 2 Einfache SPARQL-Anfragen
- 3 Komplexe Graph-Muster in SPARQL
- 4 Filter in SPARQL
- 5 Ausgabeformate in SPARQL
- 6 Modifikatoren in SPARQL
- 7 Zusammenfassung und Ausblick**

Vorgestellte SPARQL-Merkmale im Überblick

Grundstruktur

PREFIX

WHERE

Ausgabeformate

SELECT

CONSTRUCT

ASK

DESCRIBE

Graph-Muster

Einfache Graph-Muster

{...}

OPTIONAL

UNION

Filter

BOUND

isURI

isBLANK

isLITERAL

STR

LANG

DATATYPE

sameTERM

langMATCHES

REGEX

Modifikatoren

ORDER BY

LIMIT

OFFSET

DISTINCT

Offene Fragen

- Wie genau ist die Semantik von SPARQL definiert?
- Wie schwer ist die vollständige Umsetzung von SPARQL? Implementationen?
- Wie kann man Anfragen an RDF Schema oder OWL stellen?

⇒ nächste Session

Semantik von SPARQL

Sebastian Rudolph

Institut AIFB · Universität Karlsruhe

Grundlagen Semantic Web (WS09/10)

Seminar für Computerlinguistik, Universität Heidelberg

<http://semantic-web-grundlagen.de>

Foliensatz von M. Krötzsch. Die nichtkommerzielle Vervielfältigung, Verbreitung und Bearbeitung dieser Folien ist zulässig (→ Lizenzbestimmungen CC-BY-NC).

- 1 Einleitung und Motivation
- 2 Umwandlung von Anfragen in SPARQL-Algebra
- 3 Rechnen mit der SPARQL-Algebra
- 4 Zusammenfassung

- ❶ Einleitung und Ausblick
- ❷ XML und URIs
- ❸ Einführung in RDF
- ❹ RDF Schema
- ❺ Logik – Grundlagen
- ❻ Semantik von RDF(S)
- ❼ OWL – Syntax und Intuition
- ❽ OWL – Semantik und Reasoning
- ❾ OWL 2 – Syntax und Semantik
- ❿ SPARQL – Syntax und Intuition
- ⓫ **Semantik von SPARQL**
- ⓬ Konjunktive Anfragen und Regelsprachen

Letzte Vorlesung: SPARQL als Anfragesprache für RDF

```
PREFIX ex: <http://example.org/>
SELECT ?buch, ?autor WHERE
{ ?buch ex:VerlegtBei <http://springer.com/Verlag> .
  ?buch ex:Preis      ?preis .
  ?buch ex:Autor      ?autor
  FILTER (?preis < 35)
} ORDER BY ?preis LIMIT 10
```

Merkmale von SPARQL:

- Einfache, optionale und alternative Graphmuster
- Filter
- Ausgabeformate (SELECT, CONSTRUCT, ...)
- Modifikatoren (ORDER BY, LIMIT, ...)

Fragestellung für diese Vorlesung:

Wie genau ist die Semantik von SPARQL definiert?

Bisher lediglich informelle Darstellung von SPARQL

- Anwender: „Welche Antworten kann ich auf meine Anfrage erwarten?“
- Entwickler: „Wie genau soll sich meine SPARQL-Implementierung verhalten?“
- Hersteller: „Ist mein Produkt bereits SPARQL-konform?“

⇒ Formale Semantik schafft (hoffentlich) Klarheit . . .

Semantik von Anfragesprachen (1)

Semantik formaler Logik (siehe Vorlesung 5):

- Modelltheoretische Semantik: Welche Interpretationen erfüllen eine Wissensbasis?
- Beweistheoretische Semantik: Welche Ableitungen aus einer Wissensbasis sind zulässig?
- ...

Semantik von Anfragesprachen (1)

Semantik formaler Logik (siehe Vorlesung 5):

- Modelltheoretische Semantik: Welche Interpretationen erfüllen eine Wissensbasis?
- Beweistheoretische Semantik: Welche Ableitungen aus einer Wissensbasis sind zulässig?
- ...

Semantik von Programmiersprachen:

- Axiomatische Semantik: Welche logischen Aussagen gelten für ein Programm?
- Operationale Semantik: Wie wirkt sich die Abbarbeitung eines Programms aus?
- Denotationelle Semantik: Wie kann ein Programm als Eingabe/Ausgabe-Funktion abstrakt dargestellt werden?

Was tun mit Anfragesprachen?

Semantik von Anfragesprachen

Semantik von Anfragesprachen:

Anfragefolgerung (*query entailment*)

- Anfrage als Beschreibung zulässiger Anfrageergebnisse
- Datenbasis als Menge logischer Annahmen (Theorie)
- Ergebnis als logische Schlussfolgerung

Bsp.: OWL DL und RDF(S) als Anfragesprachen, konjunktive Anfragen

Semantik von Anfragesprachen (2)

Semantik von Anfragesprachen:

Anfragefolgerung (*query entailment*)

- Anfrage als Beschreibung zulässiger Anfrageergebnisse
- Datenbasis als Menge logischer Annahmen (Theorie)
- Ergebnis als logische Schlussfolgerung

Bsp.: OWL DL und RDF(S) als Anfragesprachen, konjunktive Anfragen

Anfragealgebra

- Anfrage als Rechenvorschrift zur Ermittlung von Ergebnissen
- Datenbasis als Eingabe
- Ergebnis als Ausgabe

Bsp.: Relationale Algebra für SQL, SPARQL-Algebra

- 1 Einleitung und Motivation
- 2 **Umwandlung von Anfragen in SPARQL-Algebra**
- 3 Rechnen mit der SPARQL-Algebra
- 4 Zusammenfassung

Übersetzung in SPARQL-Algebra

```
{ ?buch    ex:Preis    ?preis .  
  FILTER (?preis < 15)  
  OPTIONAL  
    { ?buch    ex:Titel    ?titel . }  
  { ?buch    ex:Autor    ex:Shakespeare . } UNION  
  { ?buch    ex:Autor    ex:Marlowe . }  
}
```

Semantik einer SPARQL-Anfrage:

- 1 Umwandlung der Anfrage in einen algebraischen Ausdruck
- 2 Berechnung des Ergebnisses dieses Ausdrucks

Übersetzung in SPARQL-Algebra: *BGP*

```
{ ?buch    ex:Preis    ?preis .  
  FILTER (?preis < 15)  
  OPTIONAL  
    { ?buch    ex:Titel    ?titel . }  
  { ?buch    ex:Autor    ex:Shakespeare . } UNION  
  { ?buch    ex:Autor    ex:Marlowe . }  
}
```

Erster Schritt: **Ersetzung einfacher Graph-Muster**

- Operator *BGP*
- gleichzeitig Auflösung von abgekürzten URIs

Übersetzung in SPARQL-Algebra: *BGP*

```
{ BGP (?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    { BGP (?buch <http://eg.org/Titel> ?titel.) }  
    { BGP (?buch <http://eg.org/Autor>  
          <http://eg.org/Shakespeare>.) }  
  UNION  
    { BGP (?buch <http://eg.org/Autor>  
          <http://eg.org/Marlowe>.) }  
}
```

Erster Schritt: Ersetzung einfacher Graph-Muster

- Operator *BGP*
- gleichzeitig Auflösung von abgekürzten URIs

Übersetzung in SPARQL-Algebra: *Union*

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    {BGP(?buch <http://eg.org/Titel> ?titel.)}  
    {BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Shakespeare>.)}  
  UNION  
    {BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Marlowe>.)}  
}
```

Zweiter Schritt: Zusammenfassung alternativer Graph-Muster

- Operator *Union*
- Bezug auf an UNION angrenzende Muster (\rightsquigarrow bindet stärker als Konjunktion)
- Klammerung mehrerer Alternativen wie in Vorlesung 9 besprochen (linksassoziativ)

Übersetzung in SPARQL-Algebra: *Union*

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    {BGP(?buch <http://eg.org/Titel> ?titel.)}  
    {BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Shakespeare>.)}  
  
  UNION  
    {BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Marlowe>.)}  
}
```

Zweiter Schritt: Zusammenfassung alternativer Graph-Muster

- Operator *Union*
- Bezug auf an UNION angrenzende Muster (\rightsquigarrow bindet stärker als Konjunktion)
- Klammerung mehrerer Alternativen wie in Vorlesung 9 besprochen (linksassoziativ)

Übersetzung in SPARQL-Algebra: *Union*

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    {BGP(?buch <http://eg.org/Titel> ?titel.)}  
  Union ({BGP(?buch <http://eg.org/Autor>  
           <http://eg.org/Shakespeare>.)},  
         {BGP(?buch <http://eg.org/Autor>  
           <http://eg.org/Marlowe>.)})  
}
```

Zweiter Schritt: *Zusammenfassung alternativer Graph-Muster*

- Operator *Union*
- Bezug auf an `UNION` angrenzende Muster (\rightsquigarrow bindet stärker als Konjunktion)
- Klammerung mehrerer Alternativen wie in Vorlesung 9 besprochen (linksassoziativ)

Übersetzung in SPARQL-Algebra

$Join(M_1, M_2)$	konjunktive Verknüpfung von M_1 und M_2
$LeftJoin(M_1, M_2, F)$	optionale Verknüpfung von M_1 mit M_2 unter der Filterbedingung F
$Filter(F, M)$	Anwendung des Filterausdrucks F auf M
Z	Konstante für <i>leeren Ausdruck</i>

Übersetzung in SPARQL-Algebra

$Join(M_1, M_2)$	konjunktive Verknüpfung von M_1 und M_2
$LeftJoin(M_1, M_2, F)$	optionale Verknüpfung von M_1 mit M_2 unter der Filterbedingung F
$Filter(F, M)$	Anwendung des Filterausdrucks F auf M
Z	Konstante für <i>leeren Ausdruck</i>

Verbleibende Übersetzung schrittweise von innen nach außen:

- 1 Wähle ein innerstes gruppierendes Graph-Muster M
- 2 Entferne Filterausdrücke aus M ;
 $GF :=$ Konjunktion der Filterbedingungen
- 3 Initialisiere $G := Z$, und arbeite alle Teilausdrücke UA ab:
 - Falls $UA = \text{OPTIONAL } Filter(F, A)$: $G := LeftJoin(G, A, F)$
 - Ansonsten, falls $UA = \text{OPTIONAL } A$: $G := LeftJoin(G, A, \text{true})$
 - Sonst: $G := Join(G, UA)$
- 4 Falls GF nicht leer ist: $G := Filter(GF, G)$

Übersetzung in SPARQL-Algebra: *(Left)Join, Filter* (1)

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    {BGP(?buch <http://eg.org/Titel> ?titel.)}  
  Union({BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Shakespeare>.)},  
        {BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Marlowe>.)})  
}
```


Übersetzung in SPARQL-Algebra: *(Left)Join, Filter* (1)

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    Join(Z, BGP(?buch <http://eg.org/Titel> ?titel.))  
  Union(Join(Z, BGP(?buch <http://eg.org/Autor>  
    <http://eg.org/Shakespeare>)),  
    Join(Z, BGP(?buch <http://eg.org/Autor>  
    <http://eg.org/Marlowe>)))  
}
```

Übersetzung in SPARQL-Algebra: *(Left)Join, Filter (2)*

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    Join(Z, BGP(?buch <http://eg.org/Titel> ?titel.))  
  Union(Join(Z, BGP(?buch <http://eg.org/Autor>  
    <http://eg.org/Shakespeare>)),  
    Join(Z, BGP(?buch <http://eg.org/Autor>  
    <http://eg.org/Marlowe>)))  
}
```

Übersetzung in SPARQL-Algebra: *(Left)Join, Filter* (2)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      Join(Z, BGP(?buch <http://eg.org/Preis> ?preis.)),  
      Join(Z, BGP(?buch <http://eg.org/Titel> ?titel.)),  
      true  
    ), Union(Join(Z, BGP(?buch <http://eg.org/Autor>  
      <http://eg.org/Shakespeare>)),  
      Join(Z, BGP(?buch <http://eg.org/Autor>  
      <http://eg.org/Marlowe>)))  
  )  
)
```

Operationen zur Darstellung der Modifikatoren:

<i>OrderBy</i> (G , Sortierangaben)	Sortiere Lösungen in Ergebnisliste
<i>Distinct</i> (G)	Entferne doppelte Lösungen aus Ergebnisliste
<i>Slice</i> (G , o , l)	Beschneide Ergebnisliste auf Abschnitt der Länge l ab Position o
<i>Project</i> (G , Variablenliste)	Beschränke alle Lösungen auf die angegebenen Variablen

Die Modifikator-Operationen werden in bestimmter Reihenfolge angewandt:

- 1 $G := \text{OrderBy}(G, \text{Sortieranweisungen})$, wenn `ORDER BY` mit diesen Sortieranweisungen verwendet wurde.
- 2 $G := \text{Project}(G, \text{Variablenliste})$, wenn das Format `SELECT` mit dieser Liste ausgewählter Variablen verwendet wurde.
- 3 $G := \text{Distinct}(G)$, wenn `DISTINCT` verwendet wurde.
- 4 $G := \text{Slice}(G, o, l)$, wenn Angaben „`OFFSET o`“ und „`LIMIT l`“ gemacht wurden. Standardwerte bei fehlender Angabe sind $o = 0$ und $l = \text{Länge von } G - o$.

- 1 Einleitung und Motivation
- 2 Umwandlung von Anfragen in SPARQL-Algebra
- 3 Rechnen mit der SPARQL-Algebra**
- 4 Zusammenfassung

Wie sind die Operationen der SPARQL-Algebra definiert?

Ausgabe:

- „Ergebnistabelle“ (Formatierung hier nicht relevant)

Eingabe:

- Angefragte RDF-Datenbasis
- Teilergebnisse von Unterausdrücken
- verschiedene Parameter je nach Operation

⇒ Wie sollen „Ergebnisse“ formal dargestellt werden?

Intuition: Ergebnisse kodieren Tabellen mit Variablenbelegungen

Ergebnis:

Liste von *Lösungen* (Lösungssequenz)

↪ jede Lösung entspricht einer Tabellenzeile

SPARQL-Ergebnisse

Intuition: Ergebnisse kodieren Tabellen mit Variablenbelegungen

Ergebnis:

Liste von *Lösungen* (Lösungssequenz)

↪ jede Lösung entspricht einer Tabellenzeile

Lösung:

Partielle Abbildung (Funktion)

- Definitionsbereich (Domäne): ausgewählte Menge von Variablen
- Wertebereich: URIs \cup leere Knoten \cup RDF-Literale

↪ Ungebundene Variablen sind solche, die von einer Lösung keinen Wert zugewiesen bekommen (*partielle* Funktion).

Wofür steht der „leere Ausdruck“ Z ?

Wofür steht der „leere Ausdruck“ Z ?

- Domäne: \emptyset (keine ausgewählten Ergebnisse)
- Lösungen: genau eine (es gibt eine Funktion mit leerem Wertebereich, aber nur eine)

\rightsquigarrow „Tabellen mit einer Zeile aber keiner Spalte“

Eine partielle Funktion μ ist eine **Lösung des Ausdrucks $BGP(T)$** (T : Liste von Tripeln), falls gilt:

- 1 Domäne von μ ist genau die Menge der Variablen in T
- 2 Durch Ersetzung von leeren Knoten durch URIs, leere Knoten oder RDF-Literale kann man T in eine Liste von Tripeln T' umwandeln, so dass gilt:

Alle Tripel in $\mu(T')$ kommen im angefragten Graph vor

Ergebnis von $BGP(T)$:

Liste aller solcher Lösungen μ (Reihenfolge undefiniert)

Vereinigung von Lösungen

Zwei Lösungen μ_1 und μ_2 sind **kompatibel** wenn gilt
 $\mu_1(x) = \mu_2(x)$ für alle x , für die μ_1 und μ_2 definiert sind

Vereinigung von zwei kompatiblen Lösungen μ_1 und μ_2 :

$$\mu_1 \cup \mu_2(x) = \begin{cases} \mu_1(x) & \text{falls } x \text{ in der Domäne von } \mu_1 \text{ vorkommt} \\ \mu_2(x) & \text{falls } x \text{ in der Domäne von } \mu_2 \text{ vorkommt} \\ \text{undefiniert} & \text{in allen anderen Fällen} \end{cases}$$

\rightsquigarrow einfache Intuition: Vereinigung von zusammenpassenden
Tabellenzeilen

Definition der SPARQL-Operationen

Jetzt können wir wesentliche Operationen definieren:

- $Filter(\Psi, F) = \{\mu \mid \mu \in \Psi \text{ und } \mu(F) \text{ ist ein Ausdruck mit Ergebnis } \text{true}\}$
- $Join(\Psi_1, \Psi_2) = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Psi_1, \mu_2 \in \Psi_2, \text{ und } \mu_1 \text{ kompatibel zu } \mu_2\}$
- $Union(\Psi_1, \Psi_2) = \{\mu \mid \mu \in \Psi_1 \text{ oder } \mu \in \Psi_2\}$
- $LeftJoin(\Psi_1, \Psi_2, F) = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Psi_1, \mu_2 \in \Psi_2, \text{ und } \mu_1 \text{ kompatibel zu } \mu_2 \text{ und } \mu_1 \cup \mu_2(F) \text{ ist ein Ausdruck mit Ergebnis } \text{true}\} \cup \{\mu_1 \mid \mu_1 \in \Psi_1 \text{ und für alle } \mu_2 \in \Psi_2 \text{ gilt: entweder ist } \mu_1 \text{ nicht kompatibel zu } \mu_2 \text{ oder } \mu_1 \cup \mu_2(F) \text{ ist nicht } \text{true}\}$

Legende:

Ψ, Ψ_1, Ψ_2 – Ergebnisse, μ, μ_1, μ_2 – Lösungen, F – Filterbedingung

Beispiel

```
@prefix ex:    <http://example.org/> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet      ex:Autor    ex:Shakespeare ;
               ex:Preis    "10.50"^^xsd:decimal .
ex:Macbeth     ex:Autor    ex:Shakespeare .
ex:Tamburlaine ex:Autor    ex:Marlowe ;
               ex:Preis    "17"^^xsd:integer .
ex:DoctorFaustus ex:Autor  ex:Marlowe ;
               ex:Preis    "12"^^xsd:integer ;
               ex:Titel    "The Tragical History of Doctor Faustus" .
ex:RomeoJulia  ex:Autor    ex:Brooke ;
               ex:Preis    "9"^^xsd:integer .
```

```
{ ?buch    ex:Preis    ?preis .  FILTER (?preis < 15)
  OPTIONAL { ?buch    ex:Titel    ?titel . }
  { ?buch    ex:Autor    ex:Shakespeare . } UNION
  { ?buch    ex:Autor    ex:Marlowe . }
}
```

Beispielrechnung (1)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
              BGP(?buch <http://eg.org/Autor>  
                  <http://eg.org/Marlowe>.) )  
  )  
)
```

buch
ex:Tamburlaine
ex:DoctorFaustus

Beispielrechnung (2)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
             BGP(?buch <http://eg.org/Autor>  
                  <http://eg.org/Marlowe>.) )  
  )  
)
```

buch
ex:Macbeth
ex:Hamlet

Beispielrechnung (3)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
              BGP(?buch <http://eg.org/Autor>  
                  <http://eg.org/Marlowe>.) )  
  )  
)
```

buch
ex:Hamlet
ex:Macbeth
ex:Tamburlaine
ex:DoctorFaustus

Beispielrechnung (4)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
             BGP(?buch <http://eg.org/Autor>  
                 <http://eg.org/Marlowe>.) )  
  )  
)
```

buch	preis
ex:Hamlet	"10.50"^^xsd:decimal
ex:Tamburlaine	"17"^^xsd:integer
ex:DoctorFaustus	"12"^^xsd:integer
ex:RomeoJulia	"9"^^xsd:integer

Beispielrechnung (5)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
              BGP(?buch <http://eg.org/Autor>  
                  <http://eg.org/Marlowe>.) )  
  )  
)
```

buch	titel
ex:DoctorFaustus	"The Tragical History of Doctor Faustus"

Beispielrechnung (6)

```
Filter((?preis < 15),
  Join(
    LeftJoin(
      BGP(?buch <http://eg.org/Preis> ?preis.),
      BGP(?buch <http://eg.org/Titel> ?titel.),
      true
    ), Union(BGP(?buch <http://eg.org/Autor>
      <http://eg.org/Shakespeare>.),
      BGP(?buch <http://eg.org/Autor>
      <http://eg.org/Marlowe>.) )
  )
)
```

buch	preis	titel
ex:Hamlet	"10.50"^^xsd:decimal	
ex:Tamburlaine	"17"^^xsd:integer	
ex:DoctorFaustus	"12"^^xsd:integer	"The Tragical History ..."
ex:RomeoJulia	"9"^^xsd:integer	

Beispielrechnung (7)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
              BGP(?buch <http://eg.org/Autor>  
                  <http://eg.org/Marlowe>.) )  
  )  
)
```

buch	preis	titel
ex:Hamlet	"10.50"^^xsd:decimal	
ex:Tamburlaine	"17"^^xsd:integer	
ex:DoctorFaustus	"12"^^xsd:integer	"The Tragical History ..."

Beispielrechnung (8)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
              BGP(?buch <http://eg.org/Autor>  
                  <http://eg.org/Marlowe>.) )  
  )  
)
```

buch	preis	titel
ex:Hamlet	"10.50"^^xsd:decimal	
ex:DoctorFaustus	"12"^^xsd:integer	"The Tragical History ..."

- 1 Einleitung und Motivation
- 2 Umwandlung von Anfragen in SPARQL-Algebra
- 3 Rechnen mit der SPARQL-Algebra
- 4 Zusammenfassung**

SPARQL als Anfragesprache für RDF

- W3C-Standard (beinahe), sehr große Verbreitung
- Anfrage basierend auf Graphmuster
- Diverse Erweiterungen (Filter, Modifikatoren, Ausgabeformate)
- Spezifikation von Anfragesyntax, Ergebnisformat, Anfrageprotokoll
- Semantik durch Übersetzung in SPARQL-Algebra